TOPEX/POSIEDON AUTONOMOUS MANEUVER EXPERIMENT (TAME) IN-FLIGHT EXECUTION

T. Kia, J. Mellstrom, A. Klumpp, H.S. Lin, P. Sanatar, K. Shen, and P. Vaze*; M. Nachman**

Autonomous spacecraft are becoming increasingly important for future space missions. Missions will continue to become more ambitious scientifically, and will demand more autonomy to accomplish complex tasks in uncertain environments and in close proximity to celestial bodies. Affordability is now an additional primary driver. The call is for smaller missions with greatly reduced ground control and operation. Spacecraft with highly autonomous, goal-directed systems are required. We believe that autonomy, in addition to reducing mission operations costs, will enable science objectives not currently possible. Autonomous maneuver planning and implementation is a key to future missions.

The TOPEX/POSIEDON Autonomous Maneuver Experiment (TAME) is an experiment to provide the necessary algorithms for planning and executing attitude maneuvers and a thrusting Orbital Maintenance Maneuver (OMM) autonomously. TAME's main module is the planner, an engine that, along with its auxiliary modules and a database, will reside on an existing satellite processor. The database contains certain satellite and orbit constants, as well as tables of mission constraints. Upon receiving an OMM command, the planner requests up-to-date propulsion data from the satellite's main On-Board Computer (OBC), and then designs an attitude maneuver that does not violate any constraint, geometric, power, or thermal. Another module, the sequence generator, incorporates the generated path into an OMM sequence that avoids violating other types of constraints, such as the timing and order of commands. The resulting command sequence, which includes commands for reconfiguring and conditioning the satellite and its components, is transmitted to the OBC for execution.

This paper describes the planning and execution phases of the experiment. The paper presents results obtained from the flight experiment, and shares lessons learned. TAME was executed, in flight, for the first time November 1997. During the first flight, TAME planned a violation-free maneuver and generated an executable sequence. The second time it will, in addition, transfer the generated sequence to the OBC, but the sequence will not be executed. The third and the final time, a normal OMM will be planned, transferred to the OBC, and executed. The remaining two flight executions are planned for the spring of 1998.

* Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California 91109.

** Orbital Sciences Corporation, 20301 Century Boulevard, Germantown, Maryland 20874.

INTRODUCTION

A higher level of spacecraft autonomy is becoming increasingly important in planning future space missions[1]. Missions will continue to become more ambitious, scientifically and technically, and will demand more autonomy to accomplish complex tasks in uncertain environments and in close proximity to celestial bodies. Affordability is now an additional primary driver of autonomous missions. Sponsors are calling for smaller missions with greatly reduced ground control and operation. Spacecraft with highly autonomous, goal directed systems are required to meet these new challenges. In addition to reducing the mission operations cost, autonomous systems will enable science objectives not possible with current spacecraft architectural designs. Autonomous maneuver planning and implementation is a key technology for future missions.

This paper describes the software design and in-flight implementation of the TOPEX/POSEIDON Autonomous Maneuver Experiment[2,3] (TAME) . It describes TAME's architecture, its primary module (the *planner*), associated software, and the results of the first of three planned flight experiments. The three experiments will provide proof-of-concept technology for an important area of on-board autonomy.

TAME provides the algorithms necessary for autonomously planning attitude maneuvers that execute an Orbital Maintenance Maneuver (OMM) without violating constraints. An OMM is accomplished by pointing the spacecraft thruster in the direction of the requested velocity increment (ΔV), and thrusting until the requested ΔV is imparted. In the TOPEX case, the velocity increment is always imparted in the direction of the current orbital velocity vector, to correct for drag. Therefore ΔV direction is never commanded.

The planner and its internal database resides on an existing satellite processor, along with several auxiliary modules some of which have their own databases. The databases contain certain satellite and orbit constants as well as tables of mission constraints. Upon receiving a maneuver ΔV and other data from ground commands, TAME requests tank pressures from the satellite's On-Board Computer (OBC). TAME uses the ΔV and tank pressures for computing burn duration. The planner then generates a *maneuver plan* for turning from cruise attitude to burn attitude and back, while avoiding violations of geometric, power, and thermal constraints en route. The *sequence generator*, merges the maneuver plan with predefined *OMM templates*, yielding an *OMM sequence* that avoids violations of other types of constraints, such as the timing and order of commands. The OMM sequence, which includes the commands to reconfigure and to condition the satellite and its components, is transferred to the OBC for execution.

This technology demonstration is providing data to perform cost/benefit analysis for trades between flight- and ground-based spacecraft mission operations. It is also providing approaches for new paradigms in system architecture, ground commanding, and test and verification that are necessary for designing highly autonomous event-driven flight systems.

TOPEX/POSEIDON MISSION

The TOPEX/Poseidon Satellite, herein referred to as TOPEX (Ocean Topology Experiment) was launched on August 10, 1992, from the Kourou Space Center in French Guyana. The satellite was launched into a nominal circular orbit with an altitude of 1336 Km and an inclination of 66 degrees. The TOPEX is a remote sensing mission with the primary science objective of providing sea surface altimetry from space[4]. The TOPEX/Poseidon program is jointly sponsored by The National Aeronautics and Space Administration (NASA) and Centre National d'Etudes Spatiales (CNES). This joint U.S./French mission combines each country's space ocean research missions. The Jet Propulsion Laboratory (JPL) manages the TOPEX mission for the NASA office of Space Sciences Application. JPL is also responsible for the day to day operation of the satellite. The Toulouse Space Laboratory manages the Poseidon for CNES. TOPEX was slated for a

prime mission of three years, which was completed in September 1995. NASA has approved a three years extension to the mission.

The primary science objective of the TOPEX satellite is to provide highly accurate measurements of the sea surface elevation over all of the ocean basins. The primary science requirement is to provide geocentric measurement of the global ocean sea level accurate to ± 14 cm with a precision of ± 2.4 cm along track. These requirements necessitated a frozen orbit that provides a fixed ground track every 10 sidereal days (127 orbits)[5]. To maintain the frozen orbit, the satellite occasionally performs a small thrusting maneuver referred to as an Orbit Maintenance Maneuver (OMM) or a burn.

TAME OBJECTIVE

An OMM consists of three phases: a planning phase, an implementation phase, and an execution phase. In the planning phase, the spacecraft orbit is determined, and the required velocity correction (the maneuver ΔV) is computed. In the implementation phase, the maneuver ΔV is converted to a sequence of low-level commands (an OMM sequence), which are loaded into a flight computer. After the OMM sequence is verified, a proceed command is issued to initiate the execution phase. Traditionally, the first two phases are carried out on the ground. TAME's objective was to develop capabilities for carrying out the implementation phase autonomously, in the flight computer.

In a fully autonomous approach, all three phases would be carried out in flight without any ground intervention. Although technically feasible for most modern satellites, this level of autonomy was deemed as inappropriate for TAME. TOPEX flight computers, the OBC and the 1750A, simply do not have sufficient unused capacity to support completely autonomous maneuver functions. Since the execution phase of an OMM has long been autonomous, it was decided that the next logical step would be to convert the implementation phase to an autonomous operation. With TAME, the planning phase remains a ground activity.

Figure 1 shows the TAME process. The planning phase is carried out on the ground by a navigation team; the resulting maneuver ΔV and other data are uplinked to the spacecraft. The implementation phase is carried out in flight; the resulting OMM sequence is downlinked to the ground for verification and to the OBC for execution. The execution phase is carried out in flight following receipt of a command to proceed.

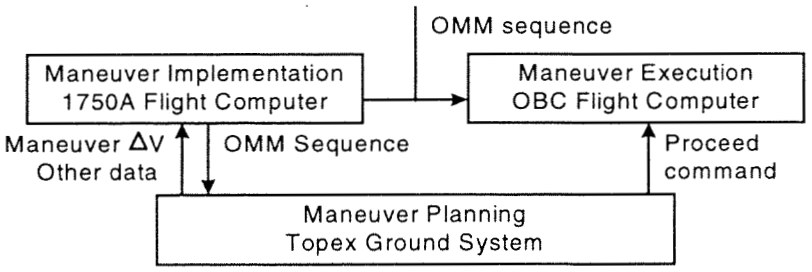

Figure 1. TOPEX/Poseidon Autonomous Maneuver process

TAME ARCHITECTURE

Figure 2 shows TAME's modular architecture. In this architecture, the OMM sequence is similar to a ground-generated OMM sequence. Thus the OBC utilizes existing maneuver routines to execute the sequence.

At the heart of the process is the planner. The planner processes OMM requests and generates the OMM sequence for execution by the OBC. Inputs to the planner are burn, window, search, and turn parameters. Maneuver ΔV is among the burn parameters. The propulsion model computes burn duration using maneuver ΔV from the planner and tank pressures from the OBC. Other inputs and other interfaces between modules are described later.

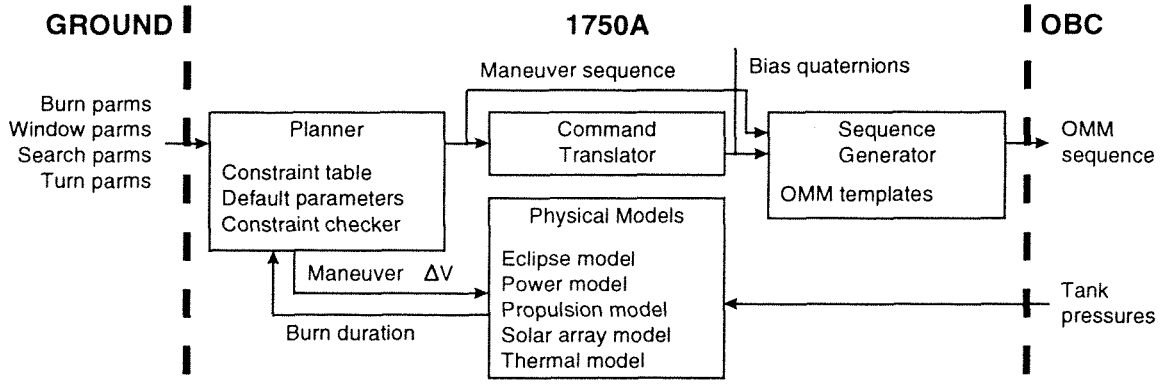

Figure 2. TAME Architecture

Communication between Flight Computers

Communication between OBC and 1750A provided another challenge to the TAME design.

Figure 3 graphically shows the proposed data transfer architecture. Communication is based on the existing TOPEX command and telemetry architecture. The 1750A output data is normally slated for transfer to ground via the Central Unit (C.U.), as part of the spacecraft telemetry. The OBC also has access to the complete telemetry via the C.U. For TAME, 1750A telemetry is modified by embedding spacecraft commands, to be executed by the OBC, in place of normal telemetry. In this experiment, the OBC software modification allows the OBC to capture, interpret, and verify the spacecraft commands from the 1750A. After extracting these spacecraft commands from 1750A telemetry, the OBC stores the TAME commands in the Absolute Time Command Buffer (ATCB) for execution at a later time. This completes the TAME maneuver implementation phase. TAME does not change the maneuver execution phase.

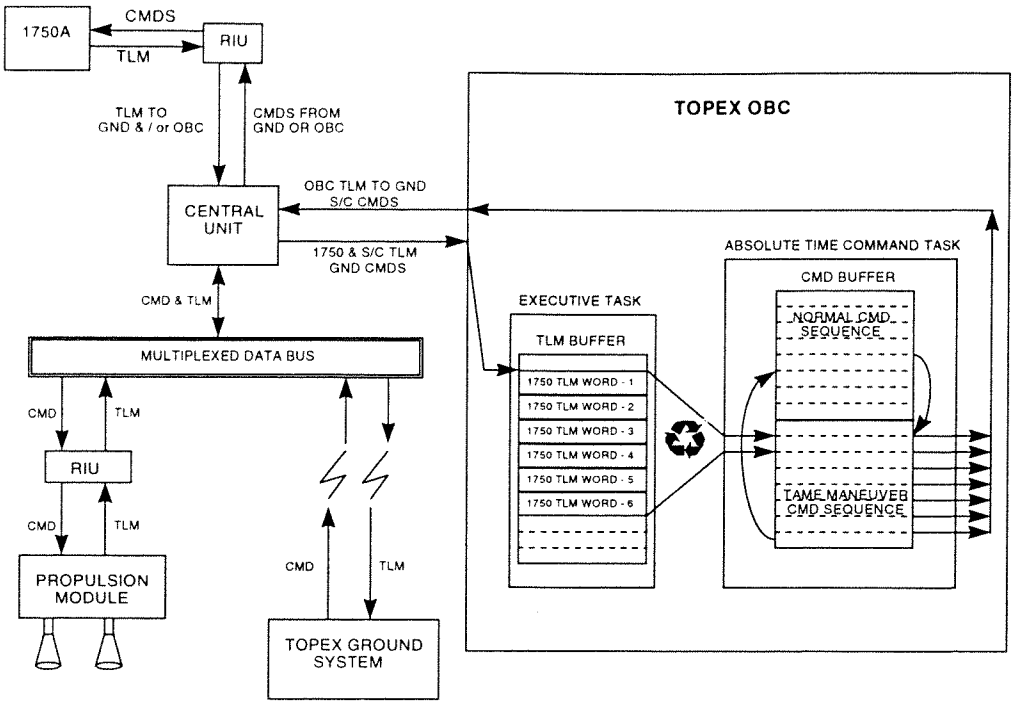

Figure 3. OBC-1750A data transfer architecture

1750A SOFTWARE ARCHITECTURE

For the purposes of the experiment, the 1750A computer in which TAME resides acts as a co-processor to the OBC. The 1750A performs all the calculations for planning the requested maneuver and generates a complete sequence to implement the planned maneuver. The OBC receives and stores the maneuver sequence of absolute timed commands from the 1750A. The sequence is then interpreted and executed.

The software on the 1750A computer is substantially original. While it was developed specifically for the TAME experiment, the underlying software architecture and command and telemetry interfaces were inherited from the pre-TAME application. The software on the OBC, on the other hand, is substantially unchanged. A patch was made to the existing software to accommodate an interactive interface with the 1750A, to receive and process a sequence generated by the 1750A, and to allow detailed ground control over execution of the autonomously generated sequence.

The inherited architecture of the 1750A software consists of a *main* program that spawns two processes referred to as the *high-* and *low-priority tasks* (see Figure 5 on page 7).

Interacting Modules

Figure 4 shows the principal TAME modules and the data that are transmitted from one module to another. Functions of the principal modules are:

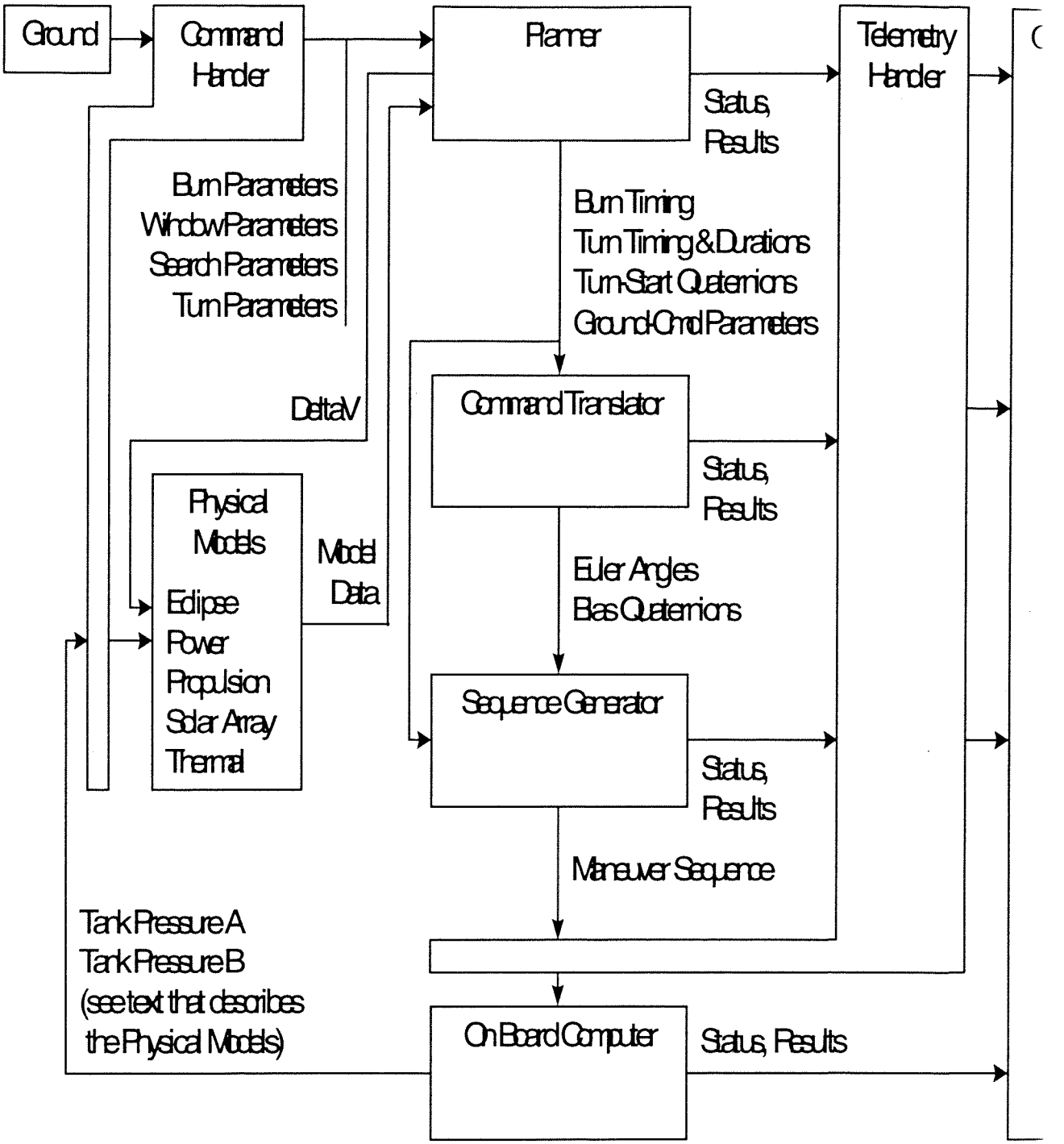

Figure 4. Flight Software Modules and Data Flow

- The ***planner*** searches for a violation-free attitude profile that turns the spacecraft from its initial attitude to the burn attitude and, after the burn, returns the spacecraft to the attitude it would have at that time had no burn been performed. Violations that would cause an attitude profile to be rejected are pointing the spacecraft's Z-axis too far from nadir,

pointing one of two trackers too close to the sun, overheating, and exceeding the battery's discharge limit. The initial and final attitudes are on a yaw-steering profile, which is based on the instantaneous geometry of the spacecraft, Earth, and Sun. As an aid in avoiding violations, the attitude profile includes two intermediate attitudes, A and B (see Figure 6 on page 9). Attitude A is along the path from the initial attitude to the burn attitude, and B is along the return path following the burn. The turns account for turn-rate limits, settling times, and spacecraft dynamic behavior. The planner supplies the command translator and sequence generator the starting and ending attitude for each turn. Attitudes are supplied as quaternions. Thus, for the four turns, the planner supplies five quaternions. The attitude during the burn is constant, so no additional quaternions are required to define the entire attitude profile.

- The ***command translator*** uses the quaternions supplied by the planner to compute a sequence of bias quaternions to be used by the sequence generator. The bias quaternions interpolate between the planner-supplied quaternions, thereby producing a sequence of uniformly spaced intermediate attitude points along any turn whose turn angle exceeds a lower limit.

- The ***sequence generator*** merges the attitude profile supplied by the planner and the command translator into a predefined sequence for preparing spacecraft hardware for the turning and thrusting maneuvers, and for restoring the coasting configuration. The predefined sequence involves pre-maneuver conditioning, such as starting catalyst-bed heaters, opening fuel latch valves, enabling propulsion module electronics, selecting thruster configuration, changing failure-detection limits, and others.

- The ***OBC*** carries out the maneuver sequence produced by the sequence generator.

- The ***physical models*** supply several types of data used for defining the planned attitude profile. Pressures in tanks A and B, and ΔV (the velocity increment to be imparted), are inputs to the propulsion model for computing burn duration. In fact, the planner requests the tank pressures from the OBC, and writes them in its specification file; the propulsion module reads the pressures from the planner spec.

- The ***command handler*** receives commands from the ground and from the OBC, and routes the commands to the designated recipient. The command handler transmits to the ground via the telemetry handler an echo of every command received from the ground, and a copy of every command received from the OBC.

- The ***telemetry handler*** packs commands, status data, the maneuver sequence, and other results in a packing buffer, and transmits the buffer to the ground. It also transmits commands and the maneuver sequence to the OBC. Telemetry data comprises every type: string, boolean, integer, float, vector, matrix, and quaternion.

Intermodule Communication

Objects communicate with one another by means of flags and modes. Each flag or mode belongs to a single object, but each can be read by any object. The command handler commonly sets flags (but not modes) belonging to other objects; other objects rarely if ever set flags or other variables that do not belong to them.

Interacting Real-Time Processes

TAME comprises five real-time processes, as shown in

Figure 5. The processes are (1) a main program, (2) a low-priority task, (3) a high-priority task, (4) a command handler, and (5) a telemetry handler.

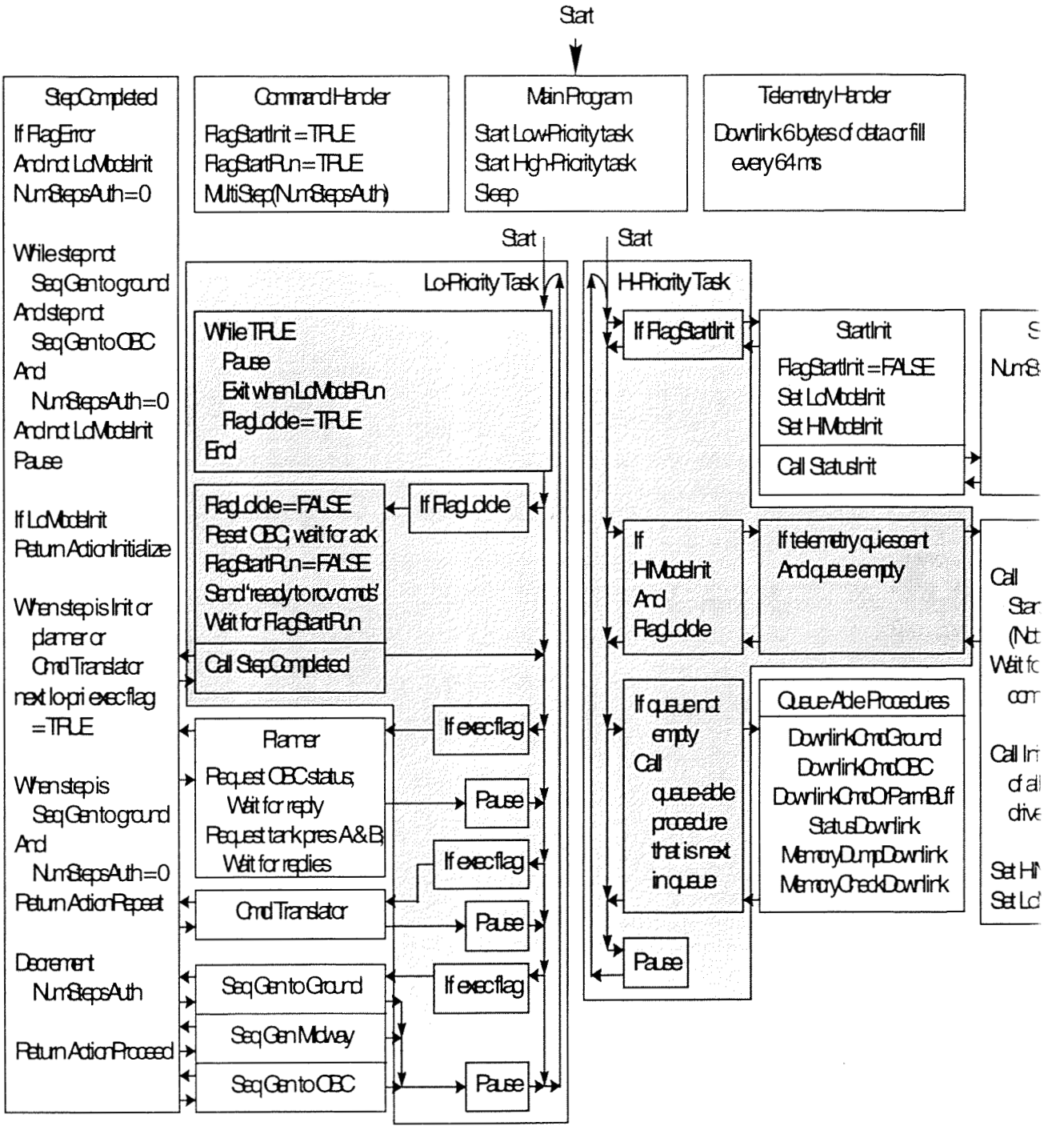

Figure 5. Flight Software's Real-Time Processes

The main program starts the two tasks, and then becomes inactive. The tasks cycle indefinitely, calling procedures in other objects but never being called. The command and telemetry handlers each provide one procedure that is driven by an interrupt stream (CommandInterface and TelemetryInterface), and numerous subprograms that are called from subprograms in the two tasks. Procedures CommandInterface and TelemetryInterface are independent of one another. The four processes that remain active are described in more detail in the following sections on the low- and high priority tasks.

The purpose of the ***low-priority task***, is to execute compute-bound modules in the background, while the high-priority task services the StartInit command and downlink requests, which require immediate attention.

Following initialization, the task enters an endless loop, polling execution flags. The value of each execution flag, TRUE or FALSE, determines whether the task calls or does not call the corresponding module, i.e., the planner, command translator, or sequence generator.

Each execution flag is set only by procedure StepCompleted (see

Figure 5). Each is initialized in its declaration to FALSE, reset to FALSE by the owning module's initialization procedure, and reset to FALSE by the procedure that is called to initiate the corresponding step.

The low-priority task's endless loop is divided into six *steps*. As the figure shows, the six steps are (1) Initialization, (2) planner execution, (3) command translator execution, (4) sequence generator telemetry to ground, (5) sequence generator midway, and (6) sequence generator telemetry to OBC. The code permits step allocations to be changed easily. The term "step" always refers to a bounded process in the low-priority task, never to one in the high-priority task.

Steps are controlled by *MultiStep* commands issued from the ground. Each of the steps ends with a call to the StepCompleted procedure, as the figure shows. The StepCompleted procedure, part of the STATUS object, receives (1) error status from each subprogram completing a step, (2) a further-steps-authorized number from the initialization process or from any MultiStep command, and (3) any StartInit command from the ground. On the basis of these data, StepCompleted sets or omits setting any execution flag for the succeeding step. On the final step, there is no execution flag for a succeeding step. Nontheless, the StepCompleted procedure returns control to the caller, so that the caller can return control to the low-priority task, which then loops endlessly waiting for a StartInit command. In any case, StepCompleted returns an action directive to the caller.

The action directive that StepCompleted returns can be any one of three: initialize, repeat, or proceed. The initialize directive is returned in response to a StartInit command, which can be issued while any step is executing or while the low-priority loop is idling in the StartInit procedure. The repeat directive is returned only when the first of the three sequence-generator steps has executed. The continue directive is returned whenever a step completes without error and further step(s) are authorized.

The content of the action directive usually does not affect the response of the recipient. The low-priority task continues its polling loop regardless of directive content. The planner and command translator return control to the low-priority task regardless of directive content. Only the sequence generator responds according to directive content. Upon completing its first step, directive content causes the sequence generator to (1) repeat the step, (2) return control to the low-priority task in response to a StartInit command, or (3) proceed to its next (second) step. Upon completing its second step, the sequence generator's responses are (2) and (3) above. Upon completing its third step, the sequence generator returns control to the low-priority task regardless of directive content, the same as the planner and command translator.

The purpose of the ***high-priority task***, is to service the StartInit command and downlink requests, which require immediate attention, while the low-priority task executes compute-bound modules in the background.

Following initialization at power-up, the task cycles endlessly polling for the StartInit flag and the presence of a queued downlink request. The StartInit flag can be set by the command handler in response to a StartInit command. Downlink requests are placed in the queue when the data for downlinking becomes available.

PLANNER IMPLEMENTATION

As mentioned earlier, the planner supplies five quaternions, shown in Figure 6, defining spacecraft attitude at the start of the OMM sequence, at inermediate point A, during the burn, at intermediate point B, and at the finish of the OMM sequence. Each of the five quaternions defines spacecraft attitude with respect to the Orbit Reference Frame (ORF). The origin of the ORF is in the spacecraft, and the ORF rotates around the center of the Earth with the Z (yaw) axis always along the nadir, the X (roll) axis forward, the Y (pitch) axis along the negative of the orbital angular momentum. The quaternions for attitudes A and B are identical, making spacecraft attitude at A and B the same with respect to the ORF, but not the same inertially because of ORF rotation.

The search has four dimensions. These are the epoch of the burn centroid and the three Euler angles that define intermediate attitudes A and B. Burn epoch is the time of the midpoint of the burn with respect to the 6 AM epoch of the orbit, expressed in seconds. The 6 AM epoch is the time when the center of the Sun is in the spacecraft's local horizontal plane, and rising due to spacecraft orbital motion.

The search for a violation-free path is performed by procedure *planner*, one of 19 subprograms (procedures and functions) in the planner module. Additional subprograms are nested in a few of the 19. The search process, including interactions of procedure planner with the other subprograms, can be described in terms of procedure planner's code. The planner search searches for a violation-free attitude path using five nested loops, as follows:

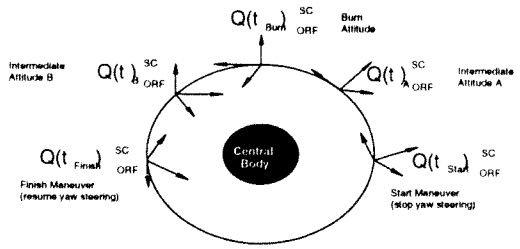

Figure 6. Attitude Profile and Intermediate Attitudes A and B

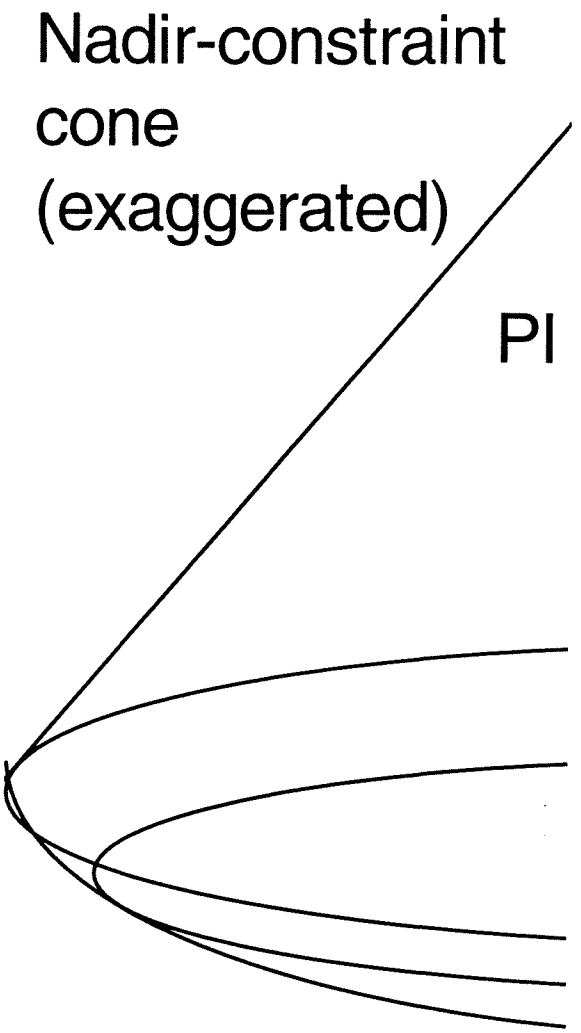

Figure 7. Search Geometry and Euler Angles Phi, Eta Psi

The two outermost loops search in the time dimension. The outer loop selects time windows within which burn epochs are permissible. The next nested loop tries burn epochs within the selected window. The three innermost loops set the Euler angles Phi, Eta, and Psi that define the intermediate attitudes A and B with respect to the ORF (see Figure 7).

All parameters used in the search are loaded by ground commands. Up to five windows and associated search parameters are received by procedure SetWndwParms. The stepsze for searching within a window and the three Euler angles are received by procedure SetSrchParms. Burn parameters, such as ΔV, are received by procedure SetBurnParms. Turn parameters, such as turn rates and times, are received by procedure SetTurnParms.

The following paragraphs describe a few of the planner's other subprograms.

Given the quaternion at intermediate attitude A, ***SlewToInfo*** computes its epoch, and the epoch and quaternion at the start of the turn that departs from the yaw steering profile. It does so in two steps. In the first step, SlewToInfo computes the epoch of arrival at intermediate attitude A by calling Turntime, which returns the duration required to turn from intermediate attitude A to the burn attitude. The desired epoch is obtained by subtracting the turn duration and settling time(s) from the epoch of arrival at the burn attitude. SlewToInfo gets the epoch of departure from the yaw steering profile, and the corresponding quaternion, by calling Merge, and subtracting the turn duration from epoch of arrival at intermediate attitude A.

Called by SlewToIno, SlewFromInfo, and Merge, ***Turntime*** computes the time required to turn from attitude Q1 to Q2.

When called by SlewToInfo, ***Merge*** computes the quaternion of departure from the yaw steering profile, and the duration required to turn from the yaw steering profile to intermediate attitude A. When called by SlewFromInfo, Merge computes the quaternion of arrival at the yaw steering profile, and the duration required to turn from intermediate attitude B to the yaw steering profile. Merge searches for the epoch numerically. It uses two search processes, as follows: The first search process depends upon whether Merge was called by SlewToInfo or SlewFromInfo. If called by SlewToInfo, Merge starts with intermediate attitude A and the epoch of arrival at intermediate attitude A. Merge then steps backward from that epoch, computing on each step the corresponding attitude of the yaw steering profile, and calling Turntime. Merge stops stepping when the turn duration returned by Turntime is less than the available time. If called by SlewFromInfo, Merge starts with intermediate attitude B and the epoch of departure from intermediate attitude B. Merge then steps forward from that epoch, computing on each step the corresponding attitude of the yaw steering profile, and calling Turntime. Merge stops stepping when the duration returned by Turntime is less than the available time. The second process is a binary search to refine the epoch of departure from or arrival at the the yaw steering profile, depending upon whether Merge was called by SlewToInfo or SlewFromInfo. In either case, Merge computes the corresponding quaternion, which it returns to the caller.

Given the quaternion at intermediate attitude B, ***SlewFromInfo*** computes its epoch, and the epoch and quaternion at the finish of the turn that arrives at the yaw steering profile. It does so in two steps. In the first step, SlewFromInfo computes the epoch of departure from intermediate attitude B by calling Turntime, which returns the duration required to turn from the burn attitude to intermediate attitude B. The desired epoch is obtained by adding the turn duration and settling time(s) to the epoch of departure from the burn attitude. SlewFromInfo gets the epoch of arrival at the yaw steering profile, and the corresponding quaternion, by calling Merge, and adding the turn duration to the epoch of departure from intermediate attitude B.

ComputeEpochsGlobal, a procedure neted in procedure planner, computes global epochs and durations not already computed.

AnalyzePath analyzes the path or attitude profile, from start to finish, in order to detect violations of any one of the constraints. AnalyzePath does the analysis in two steps. In the first step, it sets up a table of epoch points that cover the entire sequence, starting with departure from the yaw steering profile at the beginning of the first turn, and ending at the end of the settling time following arrival at the yaw steering profile at the end of the last turn. A minimum spacing separates consecutive epoch points, and a maximum number of epoch points are allowed in the duration of any one turn or coast duration. In the second step, AnalyzePath calls AnalyzePoint for each epoch point in the table. ***AnalyzePoint*** analyzes the point supplied by AnalyzePath, in order to detect violations of any one of the constraints. ***DownlinkSrchParmsAndResults*** is called repeatedly during the search process and once at the conclusion of the search process. When called during the search process,

The ***command translator*** uses the quaternions supplied by the planner to compute a sequence of bias quaternions to be used by the sequence generator. The bias quaternions interpolate between the planner-supplied quaternions, thereby producing a sequence of uniformly spaced intermediate attitude points along any turn whose turn angle exceeds a lower limit.

The method for computing the bias quaternions is chosen to match peculiarities of the Topex steering algorithms. Topex yaw steering is distinct from roll and pitch steering. The bias quaternions are intended to drive roll and pitch axes but not the yaw axis.

The bias quaternions are computed by interpolating only the roll and pitch Euler angles. The algorithm is:

1. For the roll and pitch axes, compute an intermediate Euler angle as initial + (final - initial) * i / n, where n is the number of steps, i is the step number in the range 1 .. n, and "initial" and "final" refer to the initial and final roll or pitch Euler angles. Thus, for each of the two axes, the first intermediate Euler angle is one step removed from the initial Euler angle, and the last intermediate Euler angle is identical to the final Euler angle.

2. For the yaw axis, set the intermediate Euler angle to zero.

3. Extract the bias quaternion from the product of the two direction cosine matrices that correspond to the pitch and roll intermediate Euler angles.

For generality, the code includes in the matrix product the identity matrix corresponding to the zero-value yaw Euler angle. This algorithm interpolates poorly when the pitch and roll angles are small compared to the yaw angle.

SEQUENCE GENERATOR MODULE

The ***sequence generator*** produces a sequence of stored commands (Maneuver Sequence) required to perform an OMM, similar to the sequence generated using ground-based software tools. SEQGEN merges the attitude profile supplied by the planner and the command translator into a predefined sequence for preparing spacecraft hardware and software for the turning and thrusting maneuvers, and for restoring the coasting configuration.

The architecture of the SEQGEN is described in Figure 8. The predefined sequence involves pre-maneuver conditioning, such as starting catalyst-bed heaters, opening fuel latch valves, enabling propulsion module electronics, selecting thruster configuration, changing failure-detection limits, and others.

The SEQGEN has three primary software interfaces, the command translator, the database and the telemetry formatting module. To simplify the design, the SEQGEN uses 6 hard-coded templates, which describe the pre and post maneuver commands. These templates define

the command and the relative time interval to the next command but do not define the absolute time. This allows the TAME software to produce a maneuver sequence for any desired time frame. The remaining inputs for the sequence are retrieved from the common data area. These inputs may have been uplinked from the ground or calculated by another module. The SEQGEN must sort the inputs from the CMD Translator based upon time and integrate them into the predefined sequence. The integration process also involves checking and reorganizing the command sequence based upon Spacecraft constraints. The outputs of the SEQGEN consist of data that describe a single OBC command and timetag. These data are transferred to the Telemetry Formatting module for transfer to the OBC and to the ground.

****

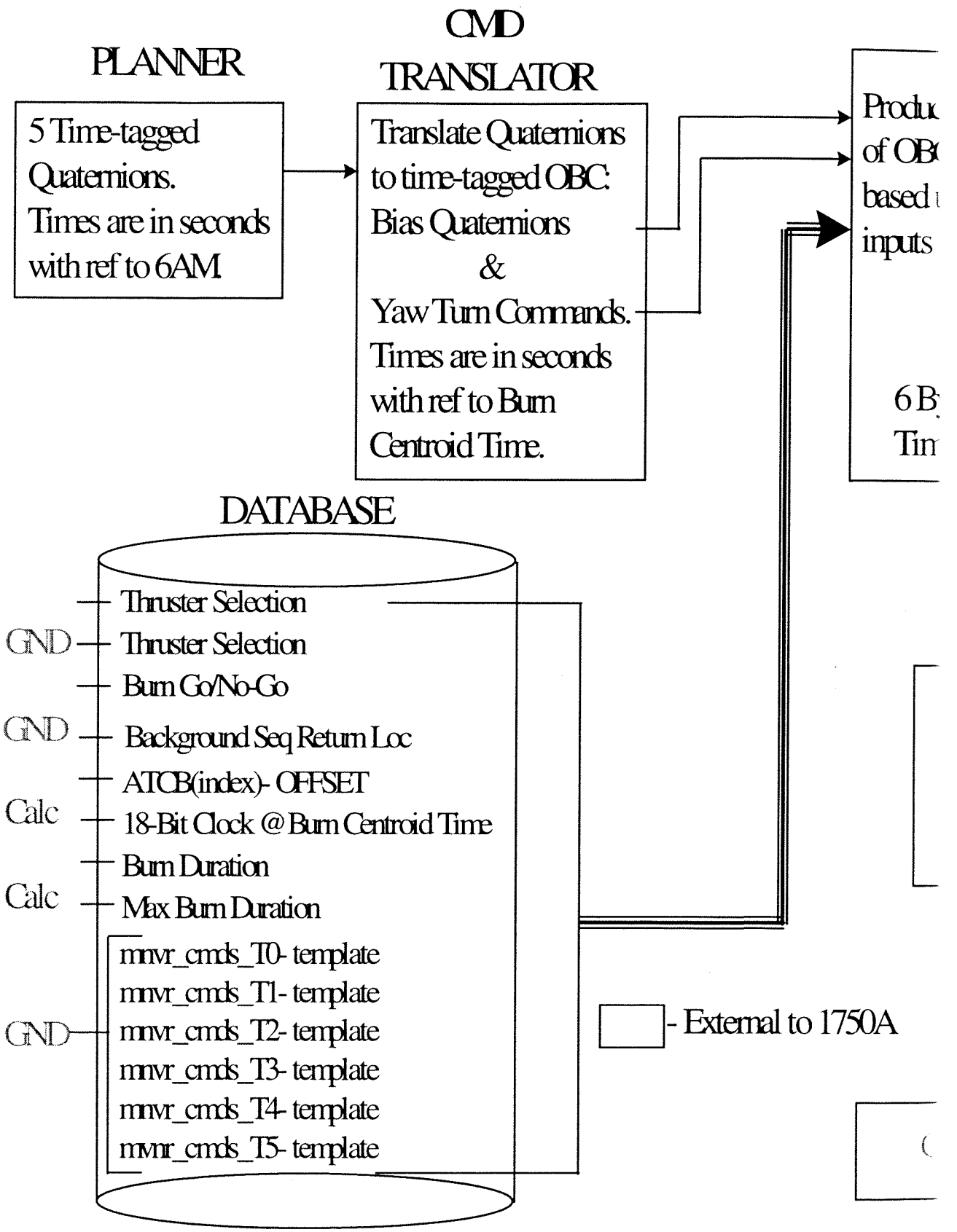
PLANNER
5 Time-tagged Quaternions.
Times are in seconds with ref to 6AM
CMD TRANSLATOR
Translate Quaternions to time-tagged OBC:
Bias Quaternions
&
Yaw Turn Commands.
Times are in seconds with ref to Burn Centroid Time.
DATABASE
Thruster Selection
GND
Thruster Selection
Burn Go/No-Go
GND
Background Seq Return Loc
ATCB(index)- OFFSET
Calc
18-Bit Clock @ Burn Centroid Time
Burn Duration
Calc
Max Burn Duration
mnvr_cmds_T0- template
mnvr_cmds_T1- template
GND
mnvr_cmds_T2- template
mnvr_cmds_T3- template
mnvr_cmds_T4- template
mnvr_cmds_T5- template
- External to 1750A

Figure 8. Sequence Generator Data Flow

The ***sequence generator*** merges the attitude profile supplied by the planner and the command translator into a predefined sequence for preparing spacecraft hardware for the turning and thrusting maneuvers, and for restoring the coasting configuration. The predefined sequence involves pre-maneuver conditioning, such as starting catalyst-bed heaters, opening fuel latch valves, enabling propulsion module electronics, selecting thruster configuration, changing failure-detection limits, and others. The sequence generator comprises two major parts, the sequence generator procedure and the compute sequence procedure.

Procedure SeqGenerator controls the steps of the low-priority task whereby the sequence is repeatedly generated and transmitted to the ground and then, when a new MultiStep command is received that authorizes further step(s), generating the sequence for transmission to the OBC. In this role, SeqGenerator is an interface between the low-priority task and the sequence generator, and also an interface between the sequence generator and the StepCompleted procedure.

The procedure consists of the following parts:

1. In an infinite loop, SeqGenerator repeatedly sets the destination to be the ground and calls procedure ComputeSequence, and then procedure StepCompleted with a FALSE error flag (signifying no error), until StepCompleted returns an action directive other than repeat.

2. SeqGenerator tests for authorization to telemeter the sequence to the OBC by calling StepCompleted with a FALSE error flag. If StepCompleted returns the action directive initialize, SeqGenerator returns control to the low-priority task rather than recomputing the sequence for transfer to the OBC.

3. SeqGenerator sets the destination to be the OBC, and calls ComputeSequence and then StepCompleted with the resulting error flag. When StepCompleted returns control to SeqGenerator, SeqGenerator returns control to the low-priority task to enable the task to run another case if and when a StartInit command is received from the ground.

TELEMETRY HANDLER ORGANIZATION

Figure 9 shows how the major components of the telemetry handler are organized. The telemetry handler provides two telemetry channels, one for subprograms of TAME's low-priority task and the other for subprograms of the high-priority task. Thus subprograms from both tasks vie for services by the telemetry handler concurrently and independently.

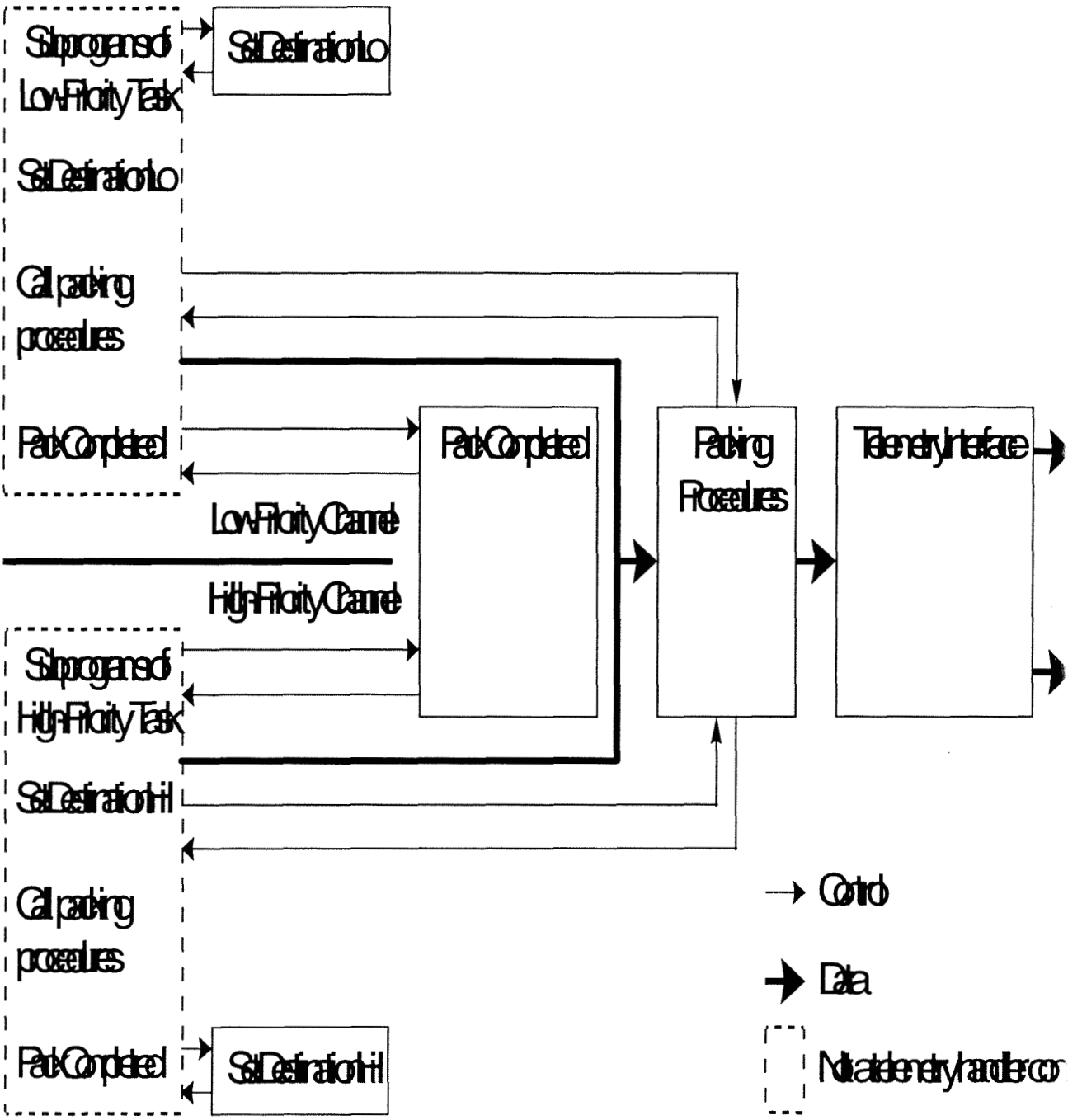

Figure 9. Telemetry Handler Organization

The two SetDestination procedures give one channel at a time access to the packing procedures. The packing procedures write telemetry data from the current channel's subprograms into a buffer (not shown in the figure). Procedure PackCompleted notifies procedure TelemetryInterface that the buffer just packed is ready to be transmitted. PackCompleted also participates in buffer and channel management. Procedure TelemetryInterface reads the data from the buffer,

and transmits all data to both the OBC and the ground. Procedure TelemetryInterface has no control connection with other components of the telemetry handler. It is invoked by a regular stream of telemetry interrupts. This is the telemetry handler's only interrupt-driven subprogram.

MULTISTEP COMMANDS

The MultiStep concept, generalizing the single-step concept, is to uplink one or more commands authorizing TAME to take a designated number of further steps. The MultiStep command includes a single integer whose value designates the number of steps that TAME is authorized to take following completion of the step underway or already completed. If the value is zero, TAME idles until it receives a new MultiStep command or a StartInit. If a MultiStep command is received and its value is greater than zero, TAME takes the number of steps designated and then returns to idling. Thus the value can always be chosen to authorize TAME to complete all processing. The default for the number of steps authorized is chosen to enable TAME to run to completion. Its value is five, one less than the total number of steps that TAME can take, because it applies after the initialization step has been taken. The default is first set when TAME is first initialized, and it is restored each time TAME is reinitialized. Thus it is never necessary to uplink a MultiStep command if it is intended for TAME to run to completion.

EXPERIMENT TEST AND VERIFICATION PLAN

The integration and test of the TAME system was driven by, the design and implementation process. Therefore, before discussing the I&T process in detail, this section will discuss the TAME design and implementation process and how it influenced the test process.

At a very high level the development can be broken down into 4 elements. These are

- TAME Algorithm Development,
- TAME Flight SW Implementation,
- Embedded System (1750A) Development, and
- OBC SW Development.

Each step is discussed in more detail below.

The TAME function, planning a constraint free attitude trajectory and creating a maneuver sequence, was originally developed in two environments. It was developed partially in MATLAB and partially in FORTRAN. FORTRAN was used wherever existing functions were deemed suitable for reuse. This occurred, for example, in the case of orbit models. New functions were implemented in MATLAB.

For the flight implementation, the algorithms were transcribed to ADA and tested on a VAX workstation. TAME Functional testing was also done in the MATLAB environment as part of the algorithm development process.

The software on the embedded system is a synthesis of code from two sources. The fundamental software architecture, interprocess communications, and external interfaces are substantially inherited. These functions are culled from the previous application code and tested in a stand-alone fashion on a 1750 target computer. The second component is the TAME Flight SW. This software, which is based on the MATLAB and FORTRAN code discussed earlier, is integrated with the inherited software and ported to the 1750.

Figure 10 depicts the TAME test environment.

At each step in the development, realistic test cases are used to verify the functionality of the planning software. These test cases fall into two categories. The first of which emulate previously conducted OMM's by restricting the planning degrees of freedom. The spacecraft attitude is required to stay nadir pointed until just before the burn and the only free variable is the burn epoch. In these test cases, the actual spacecraft telemetry is the "truth set" against which the TAME planning function is evaluated. The other class of functional test cases are those which allow off nadir pointing during the turn to the burn attitude. These test cases take advantage of the TAME attitude planning capability to "walk around" a constraint. The functionality and performance of the TAME SW in these cases is verified by analysis. These test cases were followed by project approved acceptance test cases. Table 1 summarizes the steps taken during the formal phase of the TAME tests.

FLIGHT DEMONSTRATIONS

Three in-flight tests have been planned. These tests were planned to be executed incrementally, with increasing complexity, in order to reduce the risk to the satellite and to increase confidence.

- ***Test number one*** involved loading of the 1750A software and running multiple solutions. This was executed successfully on November 17, 1997. Two OMM sequences were generated and telemetered to ground. Both sequences were for the same ΔV request, but one produced a TOPEX type solution, maintaining nadir pointing throughout the maneuver; while the second solution demonstrated TAME's capability of going off-nadir to achieve its objectives. Both generated solutions were then tested on the testbed to ensure validity. The solution was not transferred to the OBC.

- ***Test number two*** will include the OBC patch and will transfer the generated sequence to the satellite absolute time command buffer **(ATCB).** However, there are no plans to follow through and fire the thrusters. This process allows schedule flexibility, as TOPEX/Poseidon maintains strict requirements on OMM executions periods.

- Finally, ***Test number three***, will go through and complete the whole end-to-end process.

We are currently awaiting project's approval for execution of the later two tests.

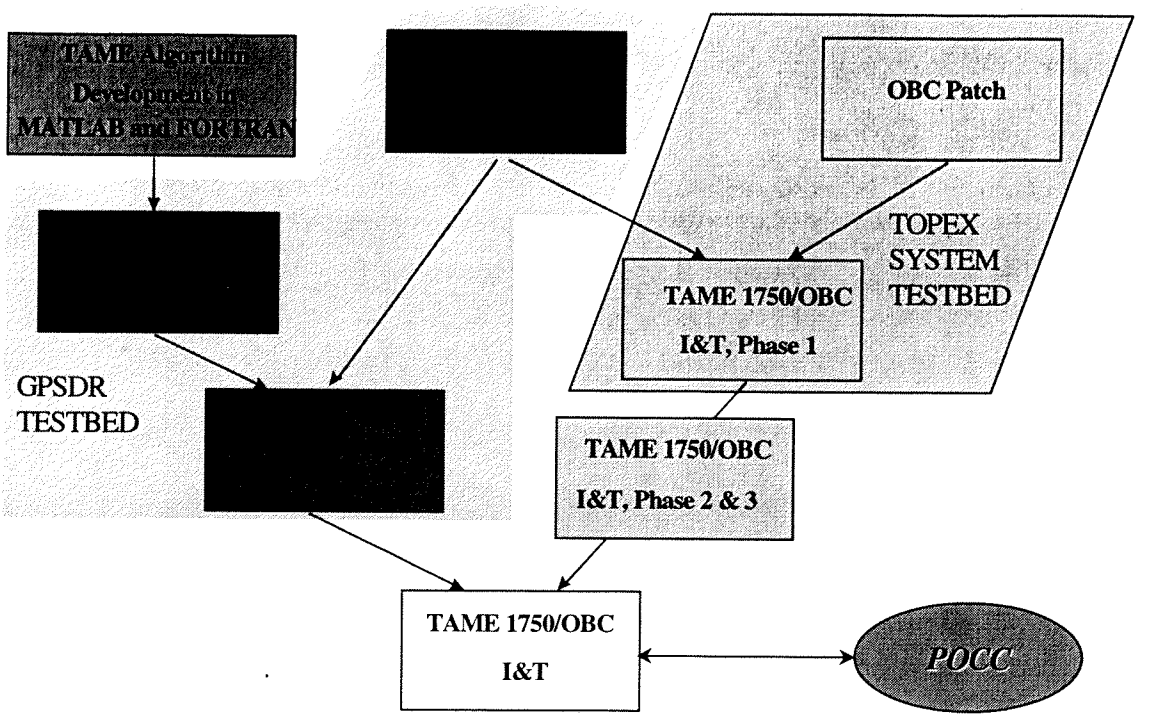

Figure 10. TAME Test Environment

RISK MITIGATION

		MATLAB	VAX	1750A	TAME	Verifica-tion	In-Flight
1	OMM5: T	X	X	X	X	Flt. Data	
2	OMM6: T	X	X			Flt. Data	
3	OMM7: T	X	X	X	X	Flt. Data	
4	OMM8: T	X	X			Flt. Data	
5	OMM9: T	X	X			Flt. Data	
7	OMM10:T	X	X	X	X	Test Bed	1,2 & 3
8	OMM10:5° Off-Nadir	X	X	X	X	Test Bed	1

9	Accep-tance Tests				X	Test Bed	

Table 1. TAME test cases.

Figure 11 thru show some of the results of the first flight experiment.

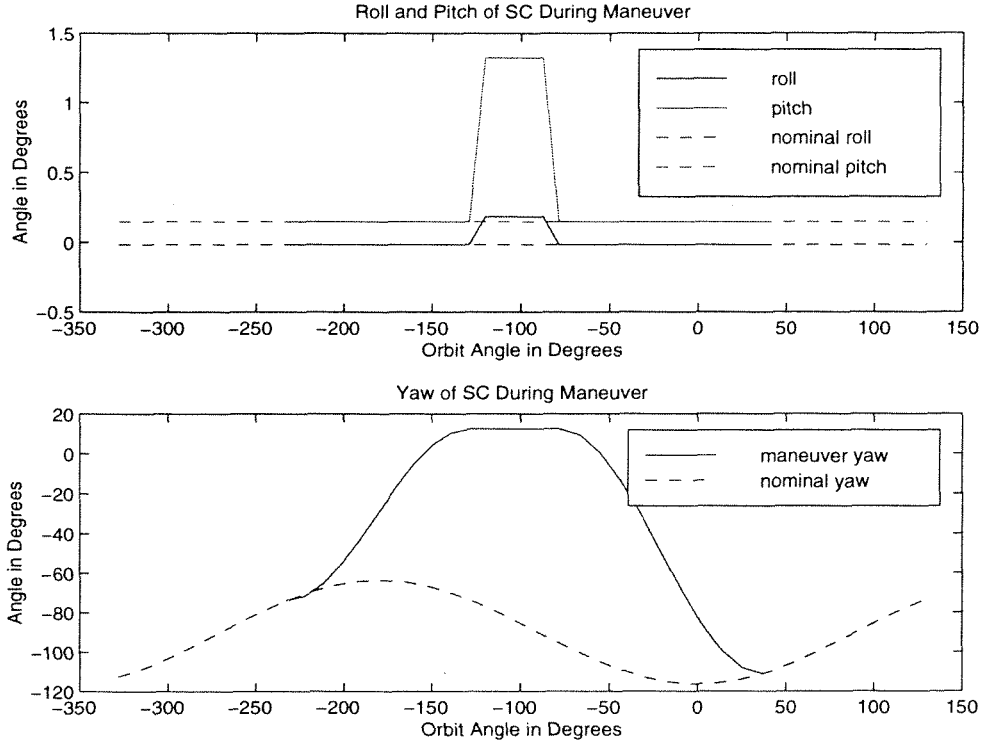

Figure 11. Yaw, Roll and Pitch Angles During TAME Maneuver

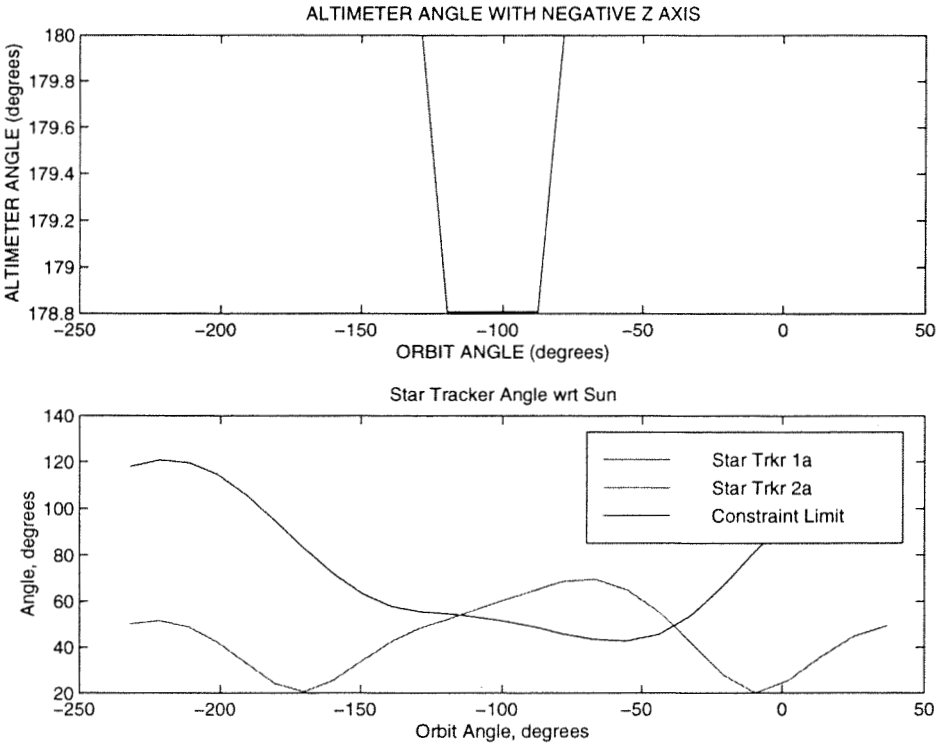

Figure 12. Star Tracker Sun Angles During TAME Maneuver

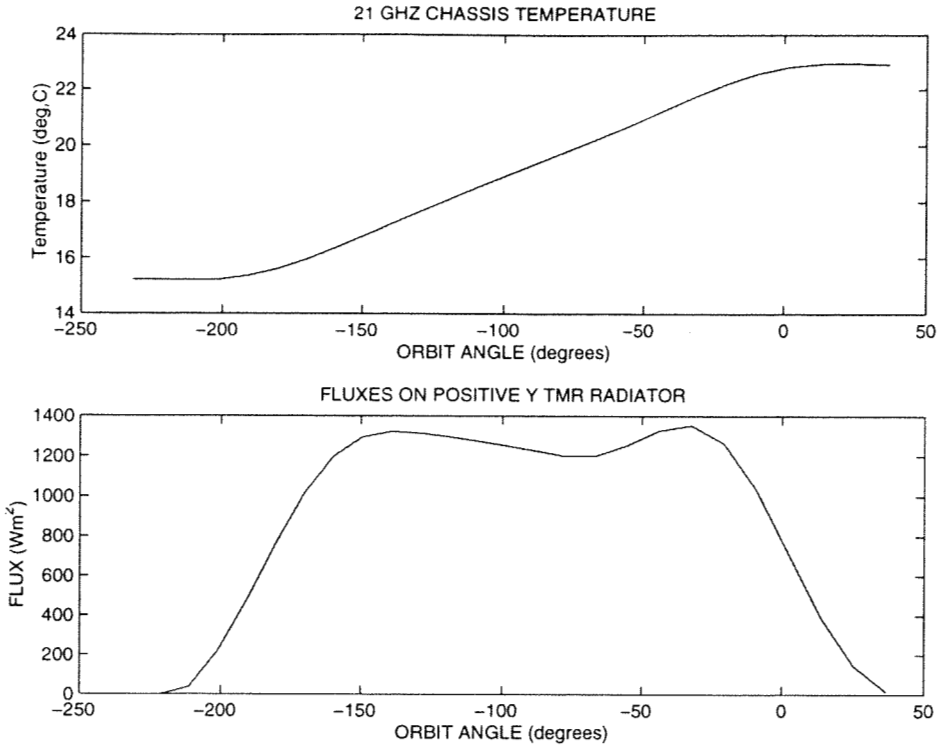

Figure 13. Primary Thermal Constraint During TAME Maneuver

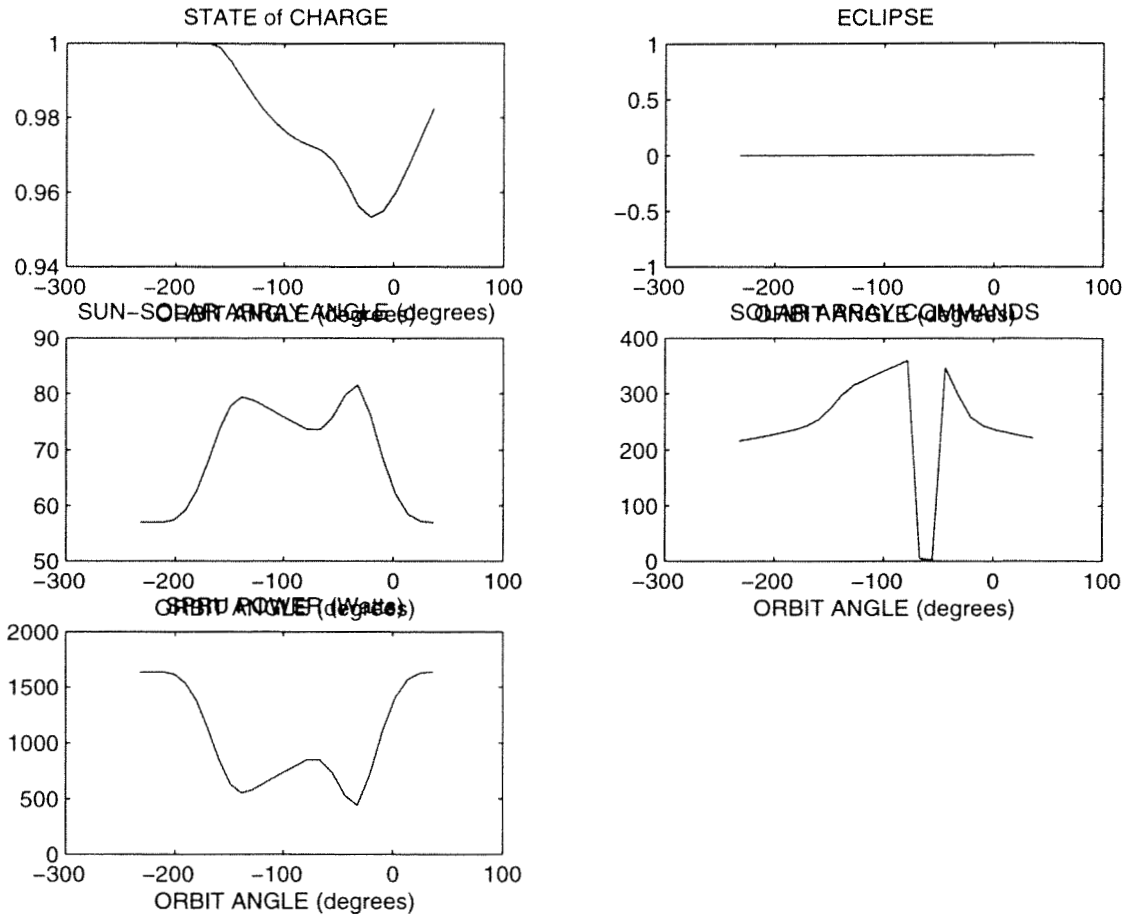

Figure 14. Battery State of Charge During TAME Maneuver

CONCLUSIONS

The feasibility for autonomous propulsive maneuver planning sequence generation was demonstrated in 1994. The 1995 effort generalized the algorithms allowing arbitrary Euler turns in place of the single axis turns and the code is being written to comply with flight software code requirements. The TOPEX Autonomous Maneuver Experiment (TAME) applies these concepts to a real operational nadir pointed orbiter.

The challenge of the TAME experiment will not be limited to the maneuver planner algorithm. Real mission constraints and oversights would have to be considered for their full impact to ensure a safe experiment.

ACKNOWLEDGMENTS

The research described in this paper was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

REFERENCES

[1] Aljabri, A. S., Kia, T., and Lai, J. Y., "Highly-Autonomous Event-Driven Spacecraft Control," *IAA International Conference on Low-Cost Planetary Missions*, Maryland, April 1994.

[2] T. Kia, A. Aljabri, R. Goddard, T. Munson, G. Kissel, H. Lin, P. Vaze, "TOPEX/POSEIDON Autonomous Maneuver Experiment (TAME)," 19th Annual AAS Guidance and Control Conference, February 7-11, 1996 Breckenridge, CO.

[3] T. Kia, J. Mellstrom, A. Klumpp, T. Munson & P. Vaze, "TOPEX/POSEIDON Autonomous Maneuver Experiment (TAME) Design and Implementation," 20th Annual AAS Guidance and Control Conference, February 5-9, 1997 Breckenridge, CO.

[4] Dennehy, C. J., Ha, K., Welch, R. and Kia, T. "On-board Attitude Determination for the TOPEX Satellite," AIAA Guidance, Navigation and Control Conference, Boston, MA, August 1989.

[5] Bhat, R. S., Frauenhlz, R. B. and Canneli, P. E. "TOPEX/POSEIDON Orbit Maintenance Maneuver Design," AAS/AIAA Astrodynamics Specialist Conference, Stowe, Vermont, August 1989.

MATERIAL NOT INCLUDED IN PAPER

Do:

- Remove references to the epemeris.
- Redraw Figure 1 because it says getting tank temperature instead of pressure, and other things? It doesn't seem to fit the associated text.
- Redraw Figure 2, including removing references to the ephemeris, and fixing other items.
- Redraw Figure "TAME Search Algorithm", fixing Ta and making other corrections shown in blue book.
- Correct any uses of the term "constraint" that actually mean "constraint violation".
- Remove all paragraphs of style Normal.
- Restore left-right orientation of Maneuver-Profile- and TAME-Search figures.
- Write section on risk mitigation.
- Finish fixing figures, and their citations.
- Make text uniform in font.
- Make text uniform in columns
- Done Space fix
- Spell check
- Search for <?>

Items wrong with figure 1:

- Fixed. It's not a thruster module but a propulsion module.
- Fixed. It's not the GPS computer but the 1750A computer.
- Fixed. It's not a command file but a maneuver sequence.
- Yes. Has OBC been introduced? Is it the "TOPEX OBC" or just the OBC?
- Fixed. We don't actually consider a solar ephemeris, and BetaPrime is only input from S/C ephemeris. And BetaPrime comes from ground, not internally.
- Fixed. S/C initial state does not come from OBC.

Items wrong with figure 2:

- Planner control doesn't exist.

ERROR: syntaxerror
OFFENDING COMMAND: --nostringval--

STACK:

-mark-
5